



# Taller de Creación de juegos Retro

## La Jaquería

# Desarrollo en Mega Drive

- Historia de la Mega Drive
- Especificaciones
- Arquitectura
- SGK
- Configuración e instalación del entorno
- Visual Studio Code
- Genesis Code
- Emulador
- Hola Mundo
- Controles
- Fondos
- Sprites
- Música y Sonido
- Scroll
- Avanzado: Mapas con TILED

# Historia de la mega Drive

La Sega Mega drive o Sega Genesis (En américa) es una videoconsola con un procesador de 16 bits; fue lanzada en Japón en 1988 y posteriormente en américa (1989) y europa (1990).

Esta videoconsola, compitió con la famosa SNES(super nintendo) vendiendo más de 30 millones de consolas.



# Historia de la mega Drive

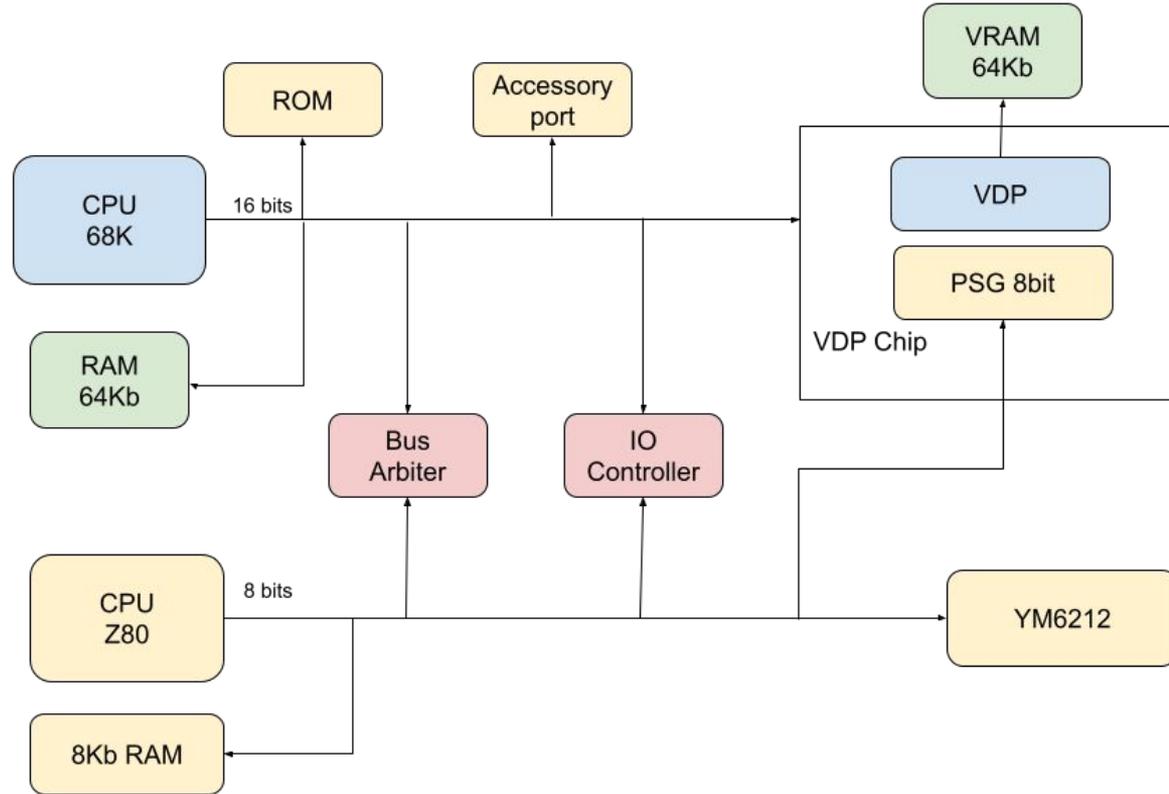
Además, de tener un gran catálogo, tenía varios periféricos como sensores de movimiento, ratón, Lector de CD, ampliaciones, etc...



# Especificaciones

Característica	Descripción
<b>CPU 1</b>	Motorola 68000 de 16 bits a 7.61Mhz (PAL), 7.67Mhz (NTSC)
<b>CPU 2</b>	Zilog Z80 de 8 bits a 3.55MHz (PAL), 3.58 Mhz (NTSC)
<b>Memoria</b>	RAM Principal: 64Kb; RAM Video: 64Kb; RAM Sonido: 8Kb
<b>Vídeo</b>	Procesador VDP con resolución de hasta 320x224 (320x240); capacidad de hasta 61 colores en pantalla (ampliable), 80 sprites máximo; 20 por línea.
<b>Sonido</b>	Chip de sonido de 6 canales Yamaha YM2612; Chip adicional de sonido de 4 canales SN76489 y sonido 8 bits
<b>Entrada/Salida</b>	Slot de cartuchos por la parte superior; Slot de expansión para la <i>Sega MegaCD</i> ; 2 conectores para controladores; salida de auriculares estereo

# Arquitectura



# Motorola 68000

```
#NO_APP
.file "main.c"
.section .rodata.str1.1,"aMS",@progbits,1
.LC0:
.string "Hello Sega!!"
.section .text.startup,"ax",@progbits
.align 2
.globl main
.type main, @function
main:
    move.l %a2,-(%sp)
    pea 13.w
    pea 10.w
    pea .LC0
    jsr VDP_drawText
    lea (12,%sp),%sp
    lea VDP_waitVSync,%a2
.L2:
    jsr (%a2)
    jsr (%a2)
    jra .L2
.size main, .-main
.ident "GCC: (GNU) 6.3.0"
```



A la hora de desarrollar para Mega Drive, se necesitaba antiguamente conocer las instrucciones para este procesador.

# Kits de Desarrollo

Antiguamente, existían kits de desarrollo para mega drive que incluían también para desarrollo de Mega-CD.

Solo estaban disponibles con licencia de SEGA y eran muy costosos.



# SGDK

SGDK (Sega Genesis Development Kit); es un conjunto de herramientas y librerías que nos va a permitir desarrollar juegos para Sega Mega Drive usando el lenguaje C.

SGDK es libre y gratuito, incluye una librería, herramientas para gestión de recursos y un compilador gcc para crear las roms. Ha sido desarrollado entre otros por Estephane Dallongeville.

Tiene Licencia MIT excepto gcc que tiene licencia GPL3.



<https://github.com/Stephane-D/SGDK>

# SGDK (Instalación)

## WINDOWS

1. Instalar Java JRE (Oracle u OpenJDK).
2. Descargar y descomprimir release:

<https://github.com/Stephane-D/SGDK/releases/tag/v1.65>

3. Crear la variable de entorno GDK apuntando a donde hemos descargado el SGDK (Opcional).

```
set GDK=<carpeta instalación SGDK>
```

# SGDK (Instalación)

## LINUX

No se puede usar por defecto SGDK en linux pero hay otros proyectos como GenDev.

1. Instalar los siguientes paquetes
  - a. texinfo
  - b. java (openjdk).
2. Descargar fichero .deb o comprimido de release:

<https://github.com/kubilus1/gendev/releases>

3. Si se ha descargado el paquete deb (solo debian), instalarlo.

```
dpkg -i <fichero.deb>
```

4. Establecer la variable de entorno GENDEV a la carpeta donde este GENDEV (por defecto /opt/gendev)

```
export GENDEV=/opt/gendev
```

# SGDK (Instalación)

## MACOS

Puedes consultar las instrucciones para MACOS en la siguiente dirección:

<https://zerasul.github.io/taller-megadrive/sgdk/instalacion/macros/>

Sin embargo, esas instrucciones pueden estar deprecadas. Recomendamos usar Docker.

# SGDK (Instalación)

## DOCKER

Este método de instalación es válido para cualquier Sistema Operativo con soporte para contenedores Docker.

1. Descargar y descomprimir SGDK de la página de Release.

<https://github.com/Stephane-D/SGDK/releases/tag/v1.65>

2. Construir la imagen:

```
docker build -t sgdk .
```

3. Para ejecutar la compilación ejecutar el siguiente comando:

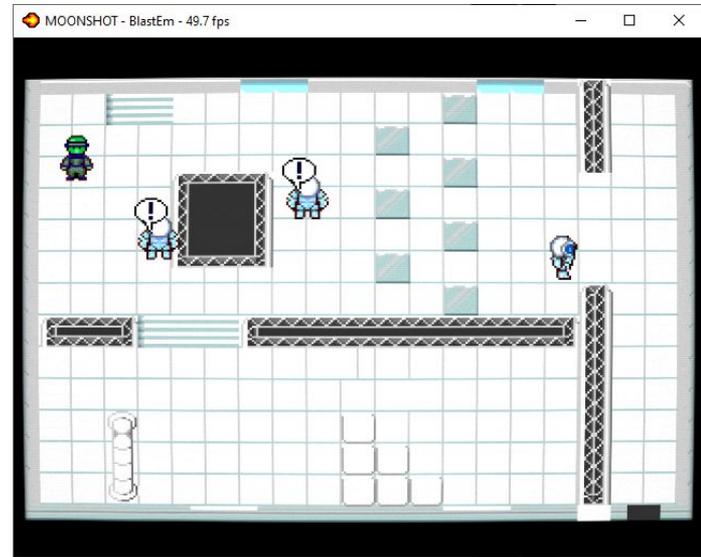
```
docker run --rm -v "$PWD":/src -u $(id -u):$(id -g) sgdk # Linux
```

```
docker run --rm -v %CD%:/src -u $(id -u):$(id -g) sgdk # Windows
```

# Emuladores

Para poder probar nuestro juego de forma más rápida, usaremos emuladores; aunque existen varios, vamos a mostrar aquí algunos como:

- Gens
- Fusion
- Blastem

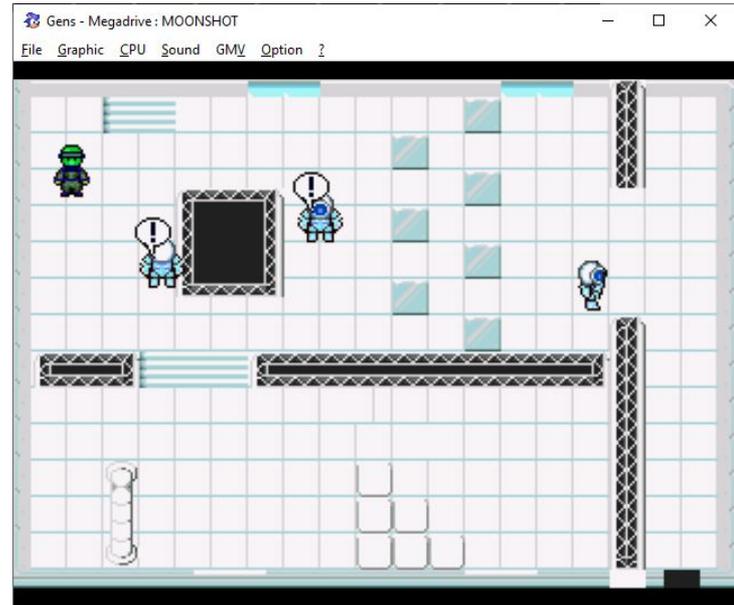


# Emuladores

## Gens

Gens; o su versión modificada, Gens Kmod(Solo windows), es un emulador de Mega Drive que nos añade una serie de herramientas para desarrollo.

- Visor de la memoria del VDP
- Visor de Sprites
- Depuración del 68K.



<https://zerasul.github.io/taller-megadrive/res/gens2.12kmod.zip>

# Emuladores

## Fusion

Kega Fusión o fusion, es un emulador que también nos ayudará a ver nuestros juegos de forma aproximada al hardware real.

[https://segaretro.org/Kega\\_Fusion](https://segaretro.org/Kega_Fusion)



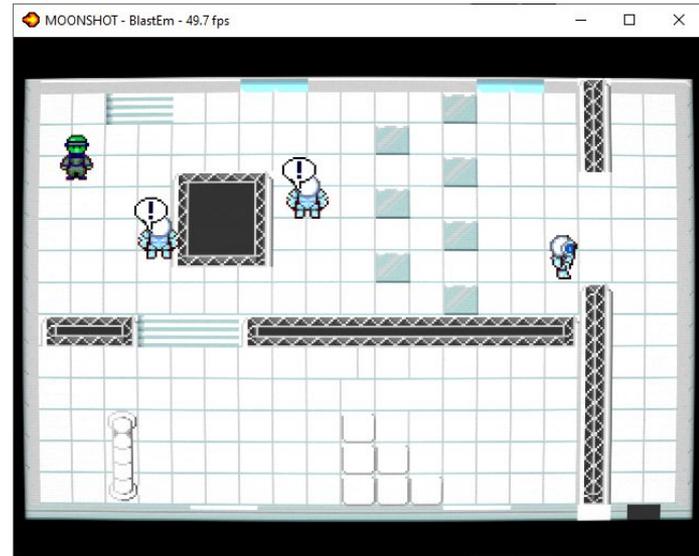
# Emuladores

## Blastem

Blastem es considerado el emulador que mejor se aproxima al hardware.

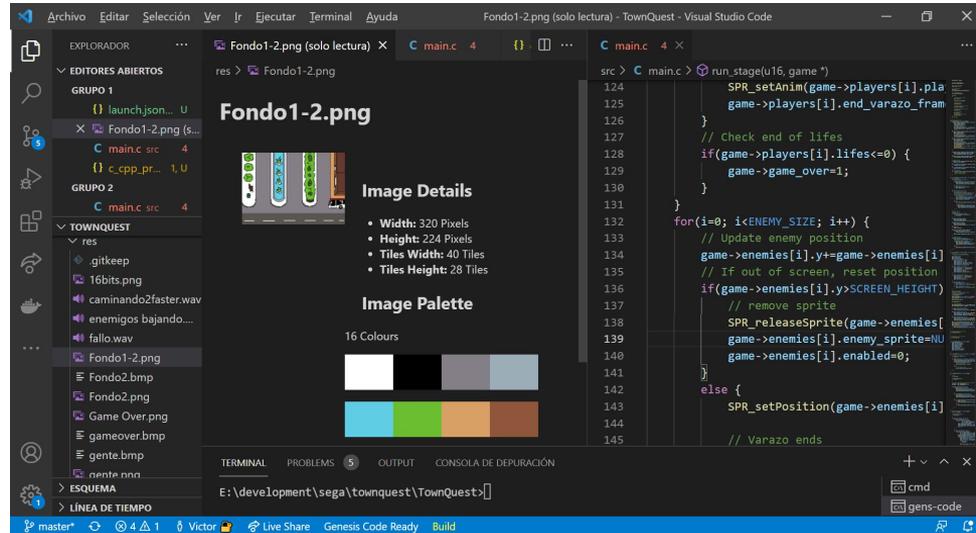
- Visor de la memoria del VDP
- Depuración del 68K.

<https://www.retrodev.com/blastem/>



# Visual Studio Code

Para ayudarnos a desarrollar, vamos a usar el editor Visual Studio Code. Este editor, nos va a ayudar a tener nuestro código más legible y nos ayudará, gracias a una extensión, a utilizar SGDK.



# Genesis Code

Genesis code, es una extensión para Visual Studio code, que nos va a ayudar a crear juegos para Sega mega Drive usando SGDK. Nos va a permitir de forma muy fácil comenzar a compilar, ejecutar y probar nuestro juego.

Además, de añadir funcionalidades al editor como autocompletado para ficheros de recursos, o un visor personalizado de imágenes.

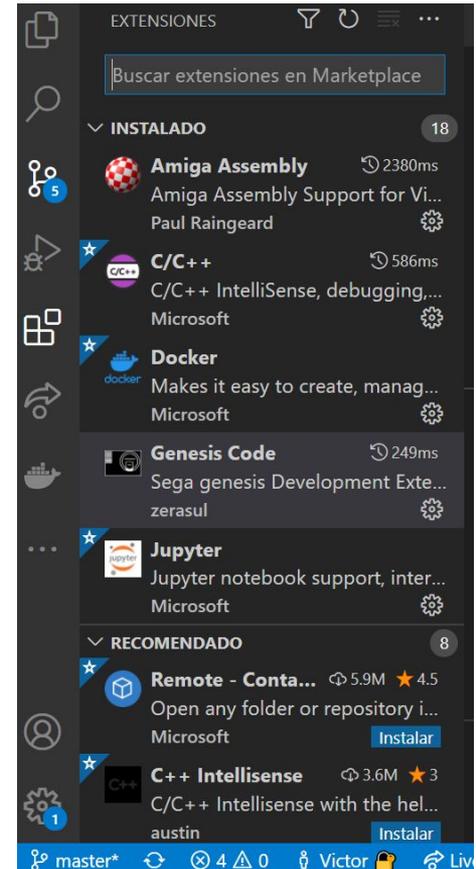


The image shows a screenshot of the Visual Studio Code extension marketplace for the 'Genesis Code' extension. The extension is by 'zerasul' and is currently at version 'v1.3.3'. It has a rating of 5 stars based on 3 reviews. The extension is described as a 'Sega genesis Development Extension'. At the bottom, there are two buttons: 'Deshabilitar' (Disable) and 'Desinstalar' (Uninstall), both with dropdown arrows, and a gear icon for settings.

**Genesis Code** v1.3.3  
zerasul | 2436 | ★★★★★ (3)  
Sega genesis Development Extension  
Deshabilitar ▼ Desinstalar ▼ ⚙️

# Genesis Code

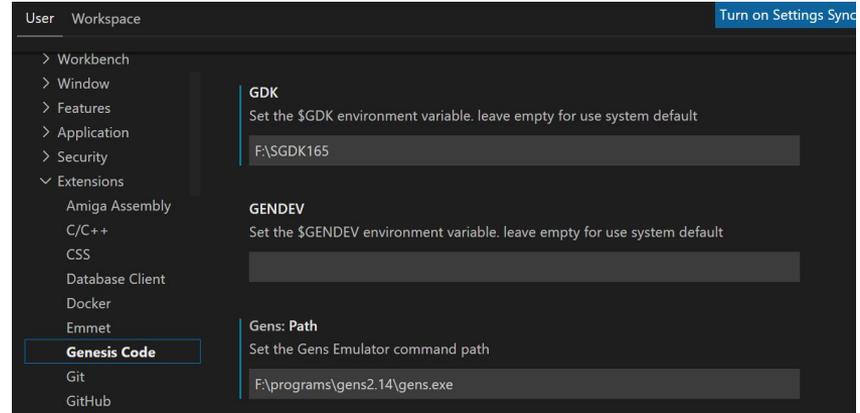
Para instalar Genesis code, acceder a la parte de extensiones de Visual Studio Code, buscarla y pulsar instalar.



# Genesis Code

Por último, puedes ver la configuración de la extensión accediendo a la misma (file->preferences->settings).

En ella podemos configurar directamente las variables de entorno a utilizar (GDK, GENDEV, MARSDEV); además de configurar el emulador a utilizar y tipo de toolchain.

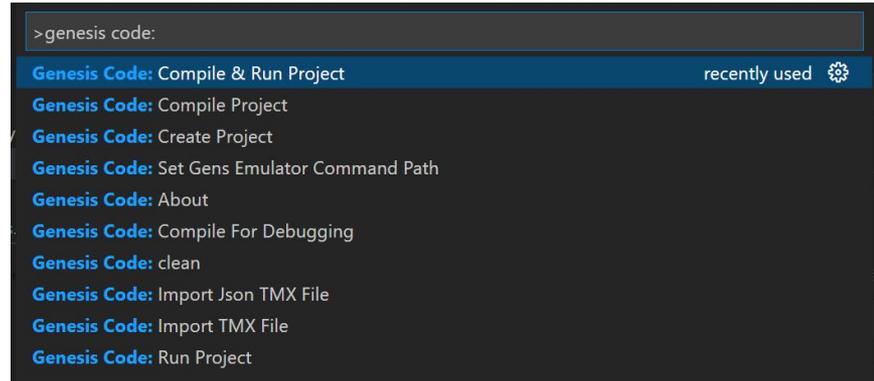


# Hola Mundo

Una vez configurado nuestro entorno, vamos a mostrar como crear un proyecto nuevo.

Para crear un proyecto, usaremos el comando “*Genesis Code: Create Project*”. Para ejecutar un comando pulsar ctrl+mayus+p.

Seleccionar la carpeta donde queremos que se cree el proyecto.



# Hola Mundo

Con el proyecto creado, veremos 3 carpetas:

- *src*: para el código fuente.
- *inc*: para los ficheros de cabecera (.h).
- *res*: para ficheros de recursos.

```
1  /**
2   * Hello World Example
3   * Created With Genesis-Code extension for Visual Studio Code
4   * Use "Genesis Code: Compile" command to compile this program.
5   **/
6  #include <genesis.h>
7
8  int main()
9  {
10     VDP_drawText("Hello Sega!!", 10,13);
11     while(1)
12     {
13         //For versions prior to SGDK 1.60 use VDP_waitVSync instead.
14         SYS_doVBlankProcess();
15     }
16     return (0);
17 }
18
```

# Hola Mundo

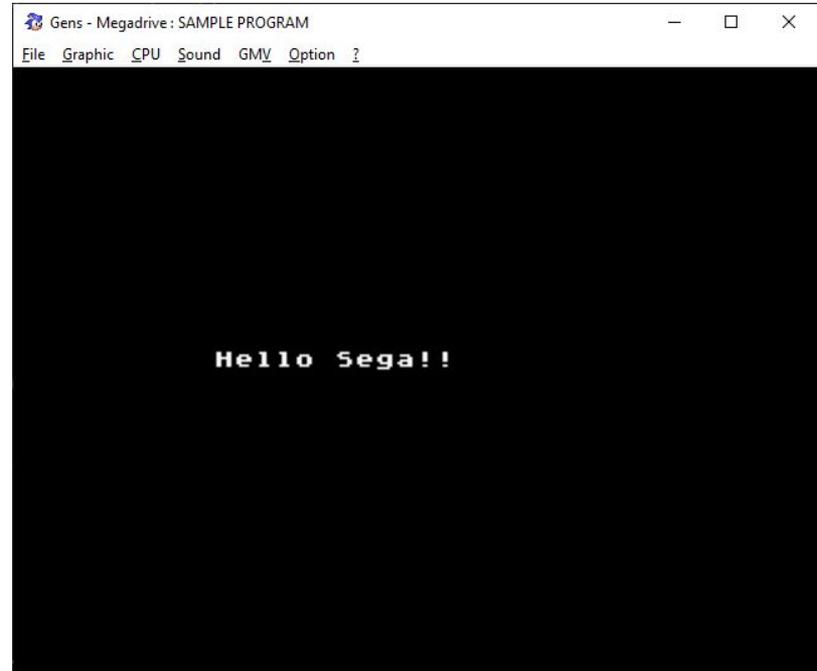
Una vez creado nuestro proyecto, usaremos el comando “Genesis Code: Compile & Run”, para compilar y ejecutar nuestro proyecto.

**NOTA:** Debes tener configurado el emulador para que se ejecute.

**NOTA2:** Por desgracia, Genesis Code aún no tiene soporte para docker. Tendrás que compilar a mano... (Estamos trabajando en la nueva versión...)

Para compilar manualmente se puede usar este comando:

```
%GDK%\bin\make -f %GDK%\makefile.gen
```



# Desarrollo de videojuegos para Mega drive

Una vez instalado y configurado nuestro entorno, ya podemos comenzar a ver cómo se crean los videojuegos para Mega Drive. Vamos a ver los siguientes puntos:

- Controles
- Fondos
- Sprites
- Música y Sonido
- Tiles
- Maps

# Controles

A la hora de crear videojuegos, tenemos que tener en cuenta los controles para que el jugador pueda interactuar.

Mega Drive, tiene por defecto 2 puertos de controlador DIN-9 o tipo Atari.



# Controles

Con SGDK, podemos tener en cuenta hasta 8 controladores, y además tener soporte para ratón.

Además, con SGDK podemos controlar los mandos de 3 o de 6 botones.



# Controles

A la hora de trabajar con controles, podemos hacerlo de dos formas:

- Síncrona
- Asíncrona.



# Controles Síncronos

Se trata de leer en cada iteración de nuestro bucle, que botones están pulsados.

```
//Se lee el estado del joystick en el puerto 1
int value = JOY_readJoypad(JOY_1);

if (value & BUTTON_UP)
```

# Controles asíncronos

Muchas veces para hacer más eficiente la lectura de los controladores, se usa la forma asíncrona; que se basa en usar Interrupciones hardware, para poder ejecutar una función al pulsar un botón.

En SGDK podemos usar la función *JOY\_setEventHandler* Para definir una función a utilizar.

```
//Establecemos el manejador de entrada  
JOY_setEventHandler( inputHandler );
```

Donde la función, debe tener la siguiente cabecera:

```
//Manejador de entrada  
void inputHandler(u16,u16,u16);
```

# Fondos

A la hora de trabajar con imágenes y gráficos en Mega Drive, tenemos que tener en cuenta las siguientes características:

- Solo existen 2 planos para dibujar.
- En cada refresco, se pintan 2 planos.
- Cada plano está compuesto por tiles (que tienen tamaño de 8x8 pixels).
- Solo se pueden almacenar 4096 tiles.
- Solo se pueden dibujar 4 paletas de 16 colores cada una.
- En total solo se pueden dibujar 61 colores en pantalla.
- En cada paleta, el primer color es el transparente (excepto el primer color de la primera paleta que será el fondo).

# Fondos

Para trabajar con recursos de nuestro juego, tenemos que definirlo y posteriormente tratarlo con la herramienta rescomp que viene incluida en el SGDK.

rescomp, lee ficheros con extensión .res con la información del recurso a tratar (imágenes de fondo, sprites, música, mapas...).

```
IMAGE fondoa "fondoa.png" BEST  
IMAGE fondob "fondob.png" BEST
```

<https://raw.githubusercontent.com/Stephane-D/SGDK/master/bin/rescomp.txt>

# Fondos

Para utilizar ficheros de imagen en SGDK, necesitaremos que estén en formato indexado (no RGB) con una paleta de colores de 16 colores; en formato bmp, png y jpeg.

Existen varios programas para ayudarnos; como puede ser GIMP, aserprite, etc...



# Fondos

Una vez incluida y relleno el fichero .res, compilamos nuestro proyecto (en genesis code “Genesis Code: compile project”) y nos generará un fichero .h con la información para incluir en nuestro código fuente.

```
#ifndef _RES_GFX_H_
#define _RES_GFX_H_

extern const Image bga_image;
extern const Image bgb_image;

#endif // _RES_GFX_H_
```

# Fondos

```
u16 ind = TILE_USERINDEX;  
VDP_drawImageEx(BG_B,&fondoa,TILE_ATTR_FULL(PAL0,FALSE,FALSE,FALSE,ind),0,0,TRUE,CPU);
```

Para dibujar uno de los fondos con una imagen, usaremos la función “*VDP\_drawImageEx*”; la cual tiene los siguientes parámetros:

- **Plano a dibujar** (BG\_A o BG\_B)
- **dirección de la imagen a dibujar** (usaremos el nombre de nuestro recurso). Recuerda incluir el fichero .h generado.
- **tile base**: Índice del tile a usar en VRAM. Se utilizará para ello, la macro *TILE\_ATTR\_FULL*.
- **Posición X**.
- **Posición Y**.
- **Cargar paleta**. TRUE para cargar paleta o FALSE para lo contrario.
- **DMA**. Usar CPU o DMA para indicar si se cargará desde CPU o usando el DMA (Direct Memory Access).

# Fondos

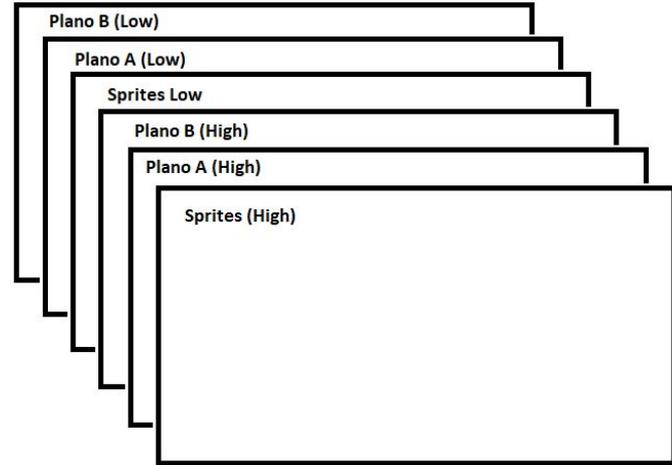
*TILE\_ATTR\_FULL*; esta macro nos permitirá definir el tile base del que dibujar. Este tile estará cargado en la VRAM del VDP. Su parámetros son:

- **Paleta**: paleta a utilizar (PAL0,PAL1,PAL2 o PAL3).
- **Prioridad**; indica si el plano se dibuja con prioridad o no.
- **flipH**: volteo horizontal.
- **flipV**: volteo vertical.
- **indice**: índice a utilizar. Normalmente se calcula a partir de *TILE\_USERINDEX*.

# Fondos

Hemos visto que se puede dibujar los fondos con o sin prioridad; esto nos permitirá dar sensación de profundidad.

**NOTA:** Existe un plano especial llamado *Window* que se utiliza para dibujar normalmente la interfaz de usuario. Suele sobrescribir en el plano A.



# Sprites

Un sprite, es un mapa de bits que normalmente es manejado por hardware específico sin necesidad de intervenir la CPU. Normalmente asociado a un objeto del juego.



# Sprites

Para usar Sprites en Mega Drive, necesitamos tener en cuenta las siguientes restricciones:

- Los Sprites tienen su propio plano
- La posición en pantalla va en pixels no en Tiles.
- Podemos tener en total 80 sprites por pantalla
- Solo se pueden tener 20 sprites por línea horizontal
- La manera de cargar un sprites es igual que los planos (en pixeles).
- Un Sprite debe ser divisible por 8 para poder cargarlo.
- El tamaño “máximo” de un sprite en SGDK es de 16x6 tiles (128 x 128 píxeles) aunque ya se permiten de más tamaño, haciendo composiciones.

# Sprites

Como cualquier recurso de nuestro juego, tenemos que añadirlo usando un fichero .res:

```
SPRITE elli_sprt "gfx/elliready.bmp" 4 4 NONE 5 BOX
```

- SPRITE: Tipo de recursos
- nombre\_variable: para referenciar
- ruta del recurso: relativa a la carpeta res.
- Ancho de cada Frame en tiles
- Alto de cada Frame en tiles
- Tipo de Compresión
- Tipo de caja de colisión (Aún no tiene uso es para futura implementación).

# Sprites

Tras definir el recurso, compilaremos para generar nuestro fichero .h que podremos añadir.

Además en nuestro código debemos hacer los siguientes pasos:

1. Inicializar el motor de sprites:

```
SPR_init();
```

2. Añadir el sprite y asignar su paleta.

```
VDP_setPalette(PAL2,elli_sprt.palette->data);  
elli = SPR_addSprite(&elli_sprt,sprx,spry,TILE_ATTR(PAL2,FALSE,FALSE,FALSE));
```

3. Actualizar el sprite en cada frame.

```
SPR_update();
```

# Sprites

La función *SPR\_addSprite*, tiene los siguientes parámetros:

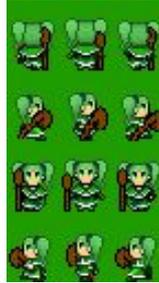
- dirección de memoria del sprite.
- posición x en pixeles.
- posición y en pixeles.
- Tile Base: Se define usando la macro *TILE\_ATTR*.

La Macro *TILE\_ATTR* tiene los siguientes parámetros:

- Paleta a utilizar.
- Flag de prioridad.
- Flag de Volteado Horizontal.
- Flag de Volteado Vertical.

# Sprites

Una parte importante a la hora de trabajar con sprites, es definir cuándo se va a utilizar una animación.



La imagen anterior es una “hoja de Sprite” o SpriteSheet que define las animaciones y frames de un sprite. Cada fila corresponde a una animación y cada columna a un frame. Por lo que la imagen anterior, corresponde a 4 animaciones de 3 frames cada una.

# Sprites

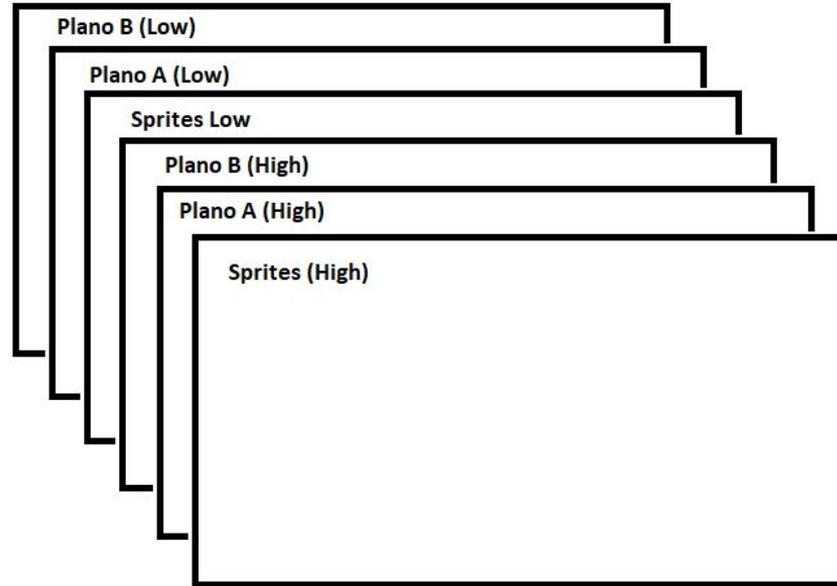
Se puede usar la función `SPR_setAnim(sprite, n° animación)` para definir qué animación utilizar.

Cada animación es una fila comenzando por la fila 0



# Sprites

Los Sprites tienen su propio plano y pueden ser dibujados con o sin prioridad. Podemos usar la función `SPR_setPriorityAttribut(sprite,TRUE|FALSE)`, para definir la prioridad.



# Matemáticas y física en SGDK

EL procesador motorola 68K, no tiene soporte para coma flotante; por lo tanto no podemos usar decimales como normalmente lo haríamos. Por ello SGDK, define dos tipos para trabajar con decimales.

Tipo	Nº Bits (signo, entero, decimal)	Rango
fix16	16 (1, 9, 6)	-512.00 a 511.00
fix32	32 (1,21,10)	- 2097152.000 a 2097151.000

# Matemáticas y física en SGDK

Además trae una serie de funciones que para ayudarnos:

Funciones	Descripción	Ejemplo
FIX16(nº)	Declara un nuevo Fix16 a partir de un número	<code>FIX16(10.5);</code>
FIX32(nº)	Declara un nuevo Fix32 a partir de un número	<code>FIX32(3456.00);</code>
intToFix16(nº)	Convierte un entero a fix16	<code>intToFix16(value_s16);</code>
intToFix32(nº)	Convierte un entero a fix32	<code>intToFix32(value_s32);</code>
fix16ToInt(nº)	Convierte un fix16 a entero (truncando)	<code>fix16ToInt(value_fix16);</code>
fix32ToInt(nº)	Convierte un fix32 a entero(truncando)	<code>fix32ToInt(value_fix32);</code>
fix16ToRoundedInt(nº)	Convierte un fix16 a entero por redondeo	<code>fix16ToRoundedInt(value_fix16);</code>
fix32ToRoundedInt(nº)	Convierte un fix32 a entero por redondeo	<code>fix32ToRoundedInt(value_fix32);</code>
fix16Add(a,b)	Realiza la suma de dos fix16.	<code>fix16Add(a,b);</code>
fix32Add(a,b)	Realiza la suma de dos fix32	<code>fix32Add(a,b);</code>

<https://zerasul.github.io/taller-megadrive/devretro/sprites/#matematicas-y-fisica-en-sgdk>

# Música y Sonido

Una parte importante de nuestro juego, será la música y sonido; gracias a la Mega Drive, podemos hacerlo usando los chips que trae:

- Yamaha YM212 de 6 canales estéreo (FM).
- Texas Instrument PSG TI6489 con sonido 8 bits.

Estos dos chips son orquestados por el procesador Z80.

**NOTA:** el procesador z80 en modo Master System no tendrá acceso al chip FM.

# Música y Sonido

Para trabajar con la música en el YM2612 o el PSG, necesitaremos un programa para poder definirla normalmente en formato vgm(XGM); estos programas llamados “trackers” nos van a permitir crear la música.



# Música y Sonido

Como cualquier recurso, tenemos que crear un fichero .res; tanto para ficheros en formato XGM o WAV como sonido.

```
XGM music "music/sor4.vgm" AUTO
```

- XGM: Tipo de recurso para música.
- nombre de la variable.
- ruta del fichero vgm
- Timing: AUTO, PAL o NTSC.

# Música y Sonido

Para definir un fichero de sonido:

```
WAV sound "snd/sound1.wav" PCM
```

- WAV: Tipo de recurso para sonido.
- nombre de la variable.
- ruta del fichero vgm
- Driver: PCM, A2ADPCM, 4PCM, XGM

Una vez añadidos los recursos, compilamos para generar el fichero .h.

# Música y sonido

Para ejecutar la música usaremos la función *SND\_startPlay\_XGM*.

```
SND_startPlay_XGM(music);
```

Donde:

- music: nombre de la variable del recurso.

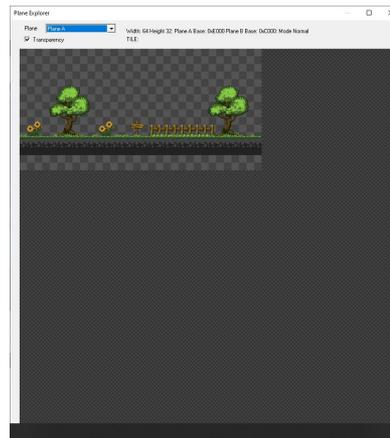
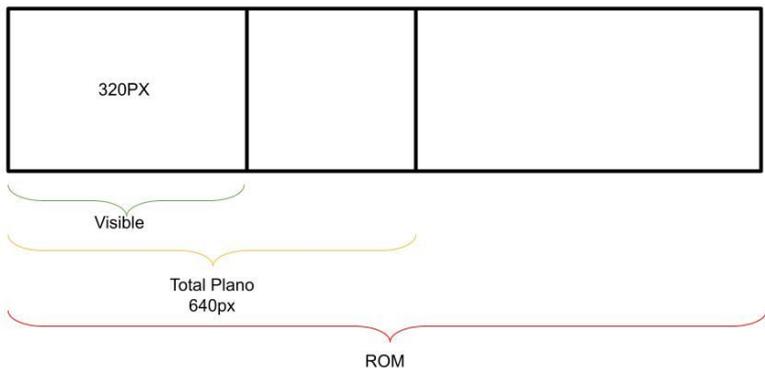
Para ejecutar sonido, tenemos que hacerlo en dos pasos:

1. Ejecutar la función *SND\_setPCM\_XGM*; con los siguientes parámetros:
  - identificador: Número identificando el recurso.
  - Variable: variable de nuestro recurso.
  - Tamaño: normalmente se usa `sizeof(variable)`.
2. Ejecutar la función *SND\_startPlayPCM\_XGM* con los siguientes parámetros:
  - ID: Identificador del recurso(debe coincidir con el de la anterior función).
  - N: N° de veces a ejecutar.
  - Canal: Canal de sonido a ejecutar. Por ejemplo `SOUND_PCM_CH2`.

# Scroll

En algunas ocasiones, nuestros fondos, van a ser mayores que la parte visible de pantalla.

Por ello, necesitaremos poder mostrar partes de las imágenes; para poder mostrar solo la parte actual.



# Scroll

SGDK, permite hacer Scroll tanto vertical como horizontalmente; por lo que se pueden extender los planos tanto como se necesiten; cargando previamente en la memoria de vídeo. Normalmente se suelen seguir los siguientes pasos:

1. Cargar toda la imagen visible
2. Conforme se avanza en una dirección hacer scroll.
3. En la parte no visible, se cargan los tiles que se necesiten por filas o columnas.

# Scroll

Para poder usar el scroll, necesitaremos usar varias funciones.

1.- Dibujar la imagen visible usando `VDP_drawImageEx`.

2.- Establecer el modo de Scroll, vertical y horizontal por plano (o línea). con la función `VDP_setScrollingMode`.

```
VDP_setScrollingMode(HSCROLL_PLANE, VSCROLL_PLANE);
```

3. *En cada iteración, cuando se quiera mover el scroll, usar la función `VDP_setHorizontalScroll` junto al plano y el offset; que está en pixeles (Positivo izquierda, negativo, derecha).*

```
VDP_setHorizontalScroll(BG_A, -offset);
```

4. *Conforme se va avanzando, se pueden ir generando tiles en la parte no visible; dando la sensación de infinito.*

```
VDP_setMapEx(BG_A, fondoa.tilemap, ...)
```

# Tilesets

Hemos visto que usando Scroll, podemos tener imágenes más grandes que los propios fondos; sin embargo, estas imágenes pueden ocupar mucho espacio en ROM. Por lo que se podrían generar dinámicamente usando Tileset.

Un Tileset es un conjunto de mapa de bits que nos van a permitir generar de forma dinámica imágenes mayores a partir de combinación de estas imágenes.

# Tilesets

En la imagen contigua, puede verse un Tileset que puede usarse para generar Mapas a partir de los elementos del mismo.

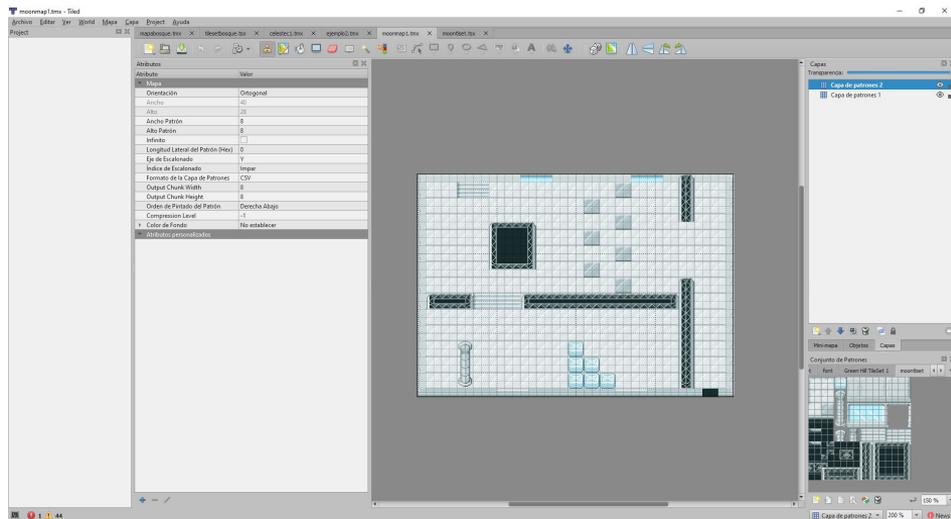
Existen muchas herramientas para poder dibujar estos mapas; nosotros usaremos TILED.



# Tilesets

TILED es una herramienta de código abierto y multiplataforma bajo licencia GPL que nos va a permitir tanto generar mapas, como poder gestionar los tilesets.

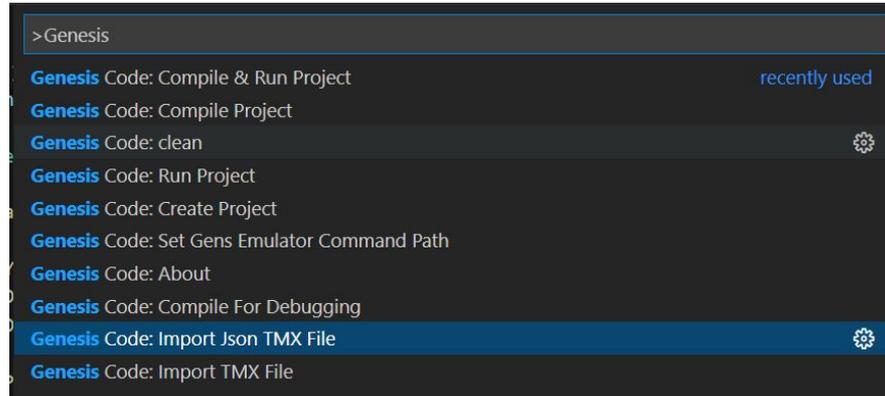
<https://www.mapeditor.org/>



# Tilesets

Con TILED, generamos cada uno de los mapas de distintos niveles (podemos generar distintas capas para después dibujarlas en distintos planos).

Posteriormente lo podemos guardar en formato TMX o Json, e importarlo en nuestro proyecto usando Genesis Code.



# Tilesets

Para comprender cómo funciona, Generamos un mapa y copiaremos las coordenadas del mapa a mano.

Una vez dibujado nuestro mapa, vamos a abrir el fichero TMX con un editor de texto.

Allí veremos una serie de números dentro de la etiqueta <data>.

**NOTA:** Guardar el mapa con formato csv y cada tile debe ocupar 8x8 pixeles.

```
<data encoding="csv">  
25,10,11,10,11,10,11,10,11,10,11,10,11,72,73,  
41,35,36,35,36,137,138,139,140,35,36,35,36,35,  
25,51,52,51,52,153,154,155,156,51,52,51,52,51,  
41,35,36,35,36,35,36,35,36,35,36,35,36,35,36,  
25,51,52,51,52,51,52,51,52,51,52,51,52,51,52,  
41,35,36,35,36,35,36,35,36,35,36,35,36,35,36,  
25,51,52,51,52,51,52,51,52,51,52,51,52,51,52,  
41,35,36,35,36,35,36,35,36,35,36,35,36,35,36,  
25,51,52,51,52,51,52,51,52,51,52,51,52,51,52.
```



# Referencias

- [Mega Drive](#)
- [Sega Retro: Technical Information](#)
- [SGDK](#)
- [Gens](#)
- [Blastem](#)
- [Kega Fusion](#)
- [Visual Studio Code](#)
- [Genesis Code](#)
- [Open Game Art](#)
- [Deflemask](#)
- [TILED](#)